Name

intro – introduction to international subroutines

Description

The internationalization package provides a convenient method of writing or converting applications so that they can operate in the application user's natural language.

The package consists of the following:

- Tools for the creation and modification of message catalogs
- An international function library, which is called *libi*
- A set of international functions available in the C library, *libc*
- An international compiler that creates language support databases from special source files
- An announcement and initialization mechanism
- A utility for converting data from one codeset to another codeset

When you use international library functions in a C program, compile it with the `-li` option to include *libi,* as shown:

```
% cc -o prog prog.c -li
```

Some of the international functions are available in the standard C library. You need not compile with the `-li` option if you use only those functions. The functions that are available in the standard C library are `setlocale`, `strftime`, `strxfrm`, and `strcoll`.

Libraries

	Internationalization Library Calls
catgetmsg	get message from a message catalog (provided for XPG–2 compatibility)
catgets	read a program message
catopen	open or close a message catalog
nl_init	set localization for internationalized program (provided for XPG–2 compatibility)
nl_langinfo	language information
nl_printf	print formatted output (provided for XPG–2 compatibility)
nl_scanf	convert formatted input (provided for XPG–2 compatibility)
printf	print formatted output
scanf	convert formatted input
vprintf	print formatted output of varargs argument list
	Standard C Library Calls
setlocale	set localization for internationalized program
strftime	convert time and date to string
strxfrm	string transformation
strcoll	string collation comparison

Header Files

i_defs.h	contains language support database structure
i_errno.h	contains error numbers and messages
langinfo.h	contains the langinfo definitions for the locale database
locale.h	contains the declarations used by the ANSI setlocale and localeconv functions
nl_types.h	contains the definitions for all the internationalization (libi) functions

See Also

iconv(1), extract(1int), gencat(1int), ic(1int), strextract(1int), strmerge(1int), trans(1int), ctype(3), setlocale(3), strcoll(3), strftime(3), strxfrm(3), catgets(3int), catopen(3int), nl_langinfo(3int), printf(3int), scanf(3int), vprintf(3int), environ(5int), lang(5int), nl_types(5int), patterns(5int)
Guide to Developing International Software

Name

catgetmsg – get message from a message catalog

Syntax

#include <nl_types.h>

nl_catd *catd*;
int *set_num, msg_num, buflen*;
char **buf*;

Description

The `catgetmsg` function has been superceded by the `catgets` function. You should use the `catgets` function to get messages from a message catalog. You might want to rewrite calls to the `catgetmsg` function so that they use the `catgets` function. The `catgetmsg` function is available for compatibility with XPG–2 conformant software and might not be available in the future. For more information on using `catgets`, see the `catgets`(3int) reference page.

The function `catgetmsg` attempts to read up to *buflen* –1 bytes of a message string into the area pointed to by *buf* . The parameter `buflen` is an integer value containing the size in bytes of *buf*. The return string is always terminated with a null byte.

The parameter *catd* is a catalog descriptor returned from an earlier call to `catopen` and identifies the message catalog containing the message set (*set_num*) and the program message (*msg_num*).

The arguments *set_num* and *msg_num* are defined as integer values for maximum portability. Where possible, you should use symbolic names for message and set numbers, rather hard-coding integer values into your source programs. If you use symbolic names, you must include the `#include` file `gencat -h` creates in all the program modules.

Return Value

If successful, `catgetmsg` returns a pointer to the message string in *buf*. Otherwise, if *catd* is invalid or if *set_num* or *msg_num* are not in the message catalog, `catgetmsg` returns a pointer to an empty (null) string.

See Also

intro(3int), gencat(1int), catopen(3int), catgets(3int), nl_types(5int)
Guide to Developing International Software

Name

catgets – read a program message

Syntax

#include <nl_types.h>

char *catgets (*catd, set_num, msg_num, s*)
nl_catd *catd*;
int *set_num, msg_num*;
char **s*;

Description

The function `catgets` attempts to read message *msg_num* in set *set_num* from the message catalog identified by *catd*. The parameter *catd* is a catalog descriptor returned from an earlier call to `catopen`. The pointer, *s*, points to a default message string. The `catgets` function returns the default message if the identified message catalog is not currently available.

The `catgets` function stores the message text it returns in an internal buffer area. This buffer area might be written over by a subsequent call to `catgets`. If you want to re-use or modify the message text, you should copy it to another location.

The arguments *set_num* and *msg_num* are defined as integer values to make programs that contain the `catgets` call portable. Where possible, you should use symbolic names for message and set numbers, instead of hard-coding integer values into your source programs. If you use symbolic names, you must include the header file that `gencat &-h` creates in all your program modules.

Examples

The following example shows using the `catgets` call to retrieve a message from a message catalog that uses symbolic names for set and message numbers:

```
nl_catd catd = catopen (messages.msf, 0)
message = catgets (catd, error_set, bad_value, "Invalid value")
```

When this call executes, `catgets` searches for the message catalog identified by the catalog descriptor stored in `catd`. The function searches for the message identified by the `bad_value` symbolic name in the set identified by the `error_set` symbolic name and stores the message text in `message`. If `catgets` cannot find the message, it returns the message Invalid value.

Return Values

If `catgets` successfully retrieves the message, it returns a pointer to an internal buffer area containing the null terminated message string. If the call is unsuccessful for any reason, *catgets* returns the default message in *s*.

See Also

intro(3int), gencat(1int), catgetmsg(3int), catopen(3int), nl_types(5int)
Guide to Developing International Software

Name

catopen, catclose – open/close a message catalog

Syntax

#include <nl_types.h>

nl_catd catopen (*name*, *oflag*)
char *name;
int oflag;

int catclose (*catd*)
nl_catd catd;

Description

The function `catopen` opens a message catalog and returns a catalog descriptor. The parameter *name* specifies the name of the message catalog to be opened. If *name* contains a slash (/), then *name* specifies a pathname for the message catalog. Otherwise, the environment variable `NLSPATH` is used with *name* substituted for *%N*. For more information, see `environ`(5int) in the *ULTRIX Reference Pages*. If `NLSPATH` does not exist in the environment, or if a message catalog cannot be opened in any of the paths specified by `NLSPATH`, the current directory is used.

The *oflag* is reserved for future use and must be set to zero (0). The results of setting this field to any other value are undefined.

The function `catclose` closes the message catalog identified by `catd`.

Restrictions

Using `catopen` causes another file descriptor to be allocated by the calling process for the duration of the `catopen` call.

Return Value

If successful, `catopen` returns a message catalog descriptor for use on subsequent calls to `catgetmsg`, `catgets` and `catclose`. If unsuccessful, `catopen` returns `(nl_catd)-1`.

The `catclose` function returns 0 if successful, otherwise -1.

See Also

intro(3int), setlocale(3), catgetmsg(3int), catgets(3int), environ(5int), nl_types(5int)
Guide to Developing International Software

nl_langinfo (3int)

Name

nl_langinfo – language information

Syntax

#include <nl_types.h>
#include <langinfo.h>

char *nl_langinfo (*item*)
nl_item *item*;

Description

The function `nl_langinfo` returns a pointer to a null-terminated string containing information relevant to a particular language or cultural area. The language is identified by the last successful call to the appropriate `setlocale` category. The categories are shown in the following table and are defined in `<langinfo.h>`.

For instance, the following example would return a pointer to the string representing the abbreviated name for the first day of the week, as defined by `setlocale` category `LC_TIME`:

```
nl_langinfo (ABDAY_1);
```

If the `setlocale` category has not been called successfully, `langinfo` data for a supported language is not available, or `item` is not defined, then `nl_langinfo` returns a pointer to an empty (null) string. In the C locale, the return value is the American English string defined in the following table:

Identifier	Meaning	C locale	Category
NOSTR	Negative response	no	LC_ALL
YESSTR	Positive response	yes	LC_ALL
D_T_FMT	Default date and time format	%a %b %d %H:%M:%S %Y	LC_TIME
D_FMT	Default date format	%m/%d/%y	LC_TIME
T_FMT	Default time format	%h:%m:%s	LC_TIME
DAY_1	Day name	Sunday	LC_TIME
DAY_2	Day name	Monday	LC_TIME
....	....	....	....
DAY_7	Day name	Saturday	LC_TIME
ABDAY_1	Abbreviated day name	Sun	LC_TIME
ABDAY_2	Abbreviated day name	Mon	LC_TIME
ABDAY_3	Abbreviated day name	Tue	LC_TIME
....	....	....	....
ABDAY_7	Abbreviated day name	Sat	LC_TIME
MON_1	Month name	January	LC_TIME
MON_2	Month name	February	LC_TIME
MON_3	Month name	March	LC_TIME
....	....	....	....

MON_12	Month name	December	LC_TIME
ABMON_1	Abbreviated month name	Jan	LC_TIME
ABMON_2	Abbreviated month name	Feb	LC_TIME
....	....	....	....
ABMON_12	Abbreviated month name	Dec	LC_TIME
RADIXCHAR	Radix character	.	LC_NUMERIC
THOUSEP	Thousands separator		LC_NUMERIC
CRNCYSTR	Currency format		LC_MONETARY
AM_STR	String for AM	AM	LC_TIME
PM_STR	String for PM	PM	LC_TIME
EXPL_STR	Lower case exponent character	e	LC_NUMERIC
EXPU_STR	Upper case exponent character	E	LC_NUMERIC

See Also

intro(3int), ic(1int), setlocale(3int), environ(5int), nl_types(5int)
Guide to Developing International Software

nl_printf(3int)

Name

nl_printf, nl_fprintf, nl_sprintf – print formatted output

Syntax

#include <stdio.h>

int nl_printf (*format* [, *arg*] ...)
char **format*;

int nl_fprintf (*stream*, *format* [, *arg*] ...)
FILE **stream*;
char **format*;

int nl_sprintf (*s*, *format* [, *arg*] ...)
char **s*, *format*;

Description

The international functions `nl_printf`, `nl_fprintf`, and `nl_sprintf` are identical to and have been superceded by the international functions `printf`, `fprintf`, and `sprintf` in a library. You should use the `printf`, `fprintf`, and `sprintf` functions when you write new calls to print formatted output in an international program. For more information on these functions, see the `printf`(3int) reference page.

You can continue to use existing calls to the `nl_printf`, `nl_fprintf`, or `nl_sprintf` international functions. These functions remain available for compatibility with XPG–2 conformant software, but may not be supported in future releases of the ULTRIX system.

The `nl_printf`, `nl_fprintf`, and `nl_sprintf` international functions are similar to the `printf` standard I/O function. (For more information about the `printf` standard I/O function, see the `printf`(3s) reference page.) The difference is that the international functions allow you to use the *I%digit$* conversion sequence in place of the % character you use in the standard I/O functions. The *digit* is a decimal digit *n* from 1 to 9. The international functions apply conversions to the *n* th argument in the argument list, rather than to the next unused argument.

You can use % conversion character in the international functions. However, you cannot mix the % conversion character with the *%digit$* conversion sequence in a single call.

You can indicate a field width or precision by an asterisk (*), instead of a digit string, in `format` strings containing the % conversion character. If you use an asterisk, you can supply an integer argument that specifies the field width or precision. In `format` strings containing the *%digit$* conversion character, you can indicate field width or precision by the sequence **digit$*. You use a decimal digit from 1 to 9 to indicate which argument contains an integer that specifies the field width or precision.

The conversion characters and their meanings are identical to `printf`.

You must use each *digit* argument at least once. The results of not using an argument are undefined.

International Environment

LC_NUMERIC If this environment is set and valid, `nl_printf` uses the international language database named in the definition to determine radix character rules.

LANG If this environment variable is set and valid `nl_printf` uses the international language database named in the definition to determine collation and character classification rules. If `LC_NUMERIC` is defined, its definition supercedes the definition of LANG.

Examples

The following example illustrates using an argument to specify field width:

```
nl_printf ("%1$d:%2$.*3$d:%4$.*3$d\n",
                              hour, min, precision, sec);
```

The format string **3$* indicates that the third argument, which is named precision, contains the integer field width specification.

To print the language independent date and time format, use the following `nl_printf` statement:

```
nl_printf (format, weekday, month, day, hour, min);
```

For United States of America use, `format` could be a pointer to the following string:

```
"%1$s,  %2$s %3$d, %4$d:%5$.2d\n"
```

This `format` string produces the following message:

```
Sunday, July 3, 10:02
```

For use in a German environment, `format` could be a pointer to the following string:

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

This `format` produces the following message:

```
Sonntag, 3. Juli, 10:02
```

See Also

intro(3int), setlocale(3), nl_scanf(3int), printf(3int), scanf(3int), printf(3s), putc(3s), scanf(3s), stdio(3s)
Guide to Developing International Software

nl_scanf(3int)

Name

nl_scanf, nl_fscanf, nl_sscanf – convert formatted input

Syntax

#include <stdio.h>

int nl_scanf (*format* [, *pointer*] ...)
char **format*;

int nl_fscanf (*stream*, *format* [, *pointer*] ...)
FILE **stream*;
char **format*;

int nl_sscanf (*s*, *format* [, *pointer*] ...)
char **s*, **format*;

Description

The international functions `nl_scanf`, `nl_fscanf`, and `nl_sscanf` are identical to and have been superceded by the international functions `scanf`, `fscanf`, and `sscanf` in *libi*. You should use the `scanf`, `fscanf`, and `sscanf` functions when you write new calls to convert formatted input in international programs. For more information on these functions, see the `scanf`(3int) reference page.

You can continue to use existing calls to the `nl_scanf`, `nl_fscanf`, or `nl_sscanf` functions. These functions remain available for compatibility with XPG–2 conformant software, but may not be supported in future releases of the ULTRIX system.

The `nl_scanf`, `nl_fscanf`, and `nl_sscanf` international functions are similar to the `scanf` standard I/O function. (For more information on the `scanf` standard I/O function, see `scanf`(3s) reference page.) The difference is that the international functions allow you to use the *%digit$* conversion character in place of the % character you use in the standard I/O functions. The *digit* is a decimal digit *n* from 1 to 9. The international functions apply conversions to the *n* th argument in the argument list, rather than to the next unused argument.

You can use the % conversion character in the international functions. However, you cannot mix the % conversion character with the *%digit$* conversion character in a single call.

International Environment

LC_NUMERIC If this environment is set and valid, `nl_scanf` uses the international language database named in the definition to determine radix character rules.

LANG If this environment variable is set and valid `nl_scanf` uses the international language database named in the definition to determine collation and character classification rules. If `LC_NUMERIC` is defined, its definition supersedes the definition of LANG.

Examples

The following shows an example of using the `nl_scanf` function:

```
nl_scanf("%2$s %1$d", integer, string)
```

If the input contains " january 9 ", the `nl_scanf` function assigns 9 to *integer* and "january" to *string* .

Return Values

These functions return either the number of items matched or EOF on end of input, along with the number of missing or invalid data items.

See Also

intro(3int), setlocale(3), strtod(3), strtol(3), nl_printf(3int), printf(3int), scanf(3int), getc(3s), printf(3s), scanf(3s)
Guide to Developing International Software

Name

printf, fprintf, sprintf – print formatted output

Syntax

#include <stdio.h>

int printf (*format* [, *arg*] ...)
char **format*;

int fprintf (*stream*, *format* [, *arg*] ...)
FILE **stream*;
char **format*;

int sprintf (*s*, *format* [, *arg*] ...)
char **s*, *format*;

Description

The international functions `printf`, `fprintf`, and `sprintf` are similar to the `printf` standard I/O functions. The difference is that the international functions allow you to use the *%digit$* conversion character in place of the % character you use in the standard I/O functions. The *digit* is a decimal digit *n* from 1 to 9. The international functions apply conversions to the *n* th argument in the argument list, rather than to the next unused argument.

You can use the % conversion character in the international functions. However, you cannot mix the % conversion character with the *%digit$* conversion character in a single call.

You can indicate a field width or precision by an asterisk (*) instead of a digit string in format strings containing the % conversion character. If you use an asterisk, you can supply an integer *arg* that specifies the field width or precision. In format strings containing the *%digit$* conversion character, you can indicate field width or precision by the sequence **digit$*. You use a decimal digit from 1 to 9 to indicate which argument contains an integer that specifies the field width or precision.

The conversion characters and their meanings are identical to `printf`.

You must use each *digit* argument at least once.

In all cases, the radix character `printf` uses is defined by the last successful call to `setlocale` category `LC_NUMERIC`. If `setlocale` category `LC_NUMERIC` has not been called successfully or if the radix character is undefined, the radix character defaults to a period (.).

International Environment

LC_NUMERIC If this environment is set and valid, `printf` uses the international language database named in the definition to determine radix character rules.

LANG If this environment variable is set and valid `printf` uses the international language database named in the definition to determine collation and character classification rules. If `LC_NUMERIC` is defined, its definition supercedes the definition of LANG.

Examples

The following example illustrates using an argument to specify field width:

```
printf ("%1$d:%2$.*3$d:%4$.*3$d\n",
                            hour, min, precision, sec);
```

The format string **3$* indicates that the third argument, which is named precision, contains the integer field width specification.

To print the language independent date and time format use the following `printf` statement:

```
printf (format, weekday, month, day, hour, min);
```

For American use, *format* could be a pointer to the following string:

```
"%1$s,  %2$s %3$d, %4$d:%5$.2d\n"
```

This string gives the following date format:

```
Sunday, July 3, 10:02
```

For use in a German environment, *format* could be a pointer to the following string:

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

This string gives the following date format:

```
Sonntag, 3. Juli, 10:02
```

Return Values

`printf` and `fprintf` return zero for success and EOF for failure. The `sprintf` subroutine returns its first argument for success and EOF for failure.

In the System V and POSIX environments, `printf`, `fprintf`, and `sprintf` return the number of characters transmitted for success. The `sprintf` function ignores the null terminator (\0) when calculating the number of characters transmitted. If an output error occurs, these routines return a negative value.

See Also

intro(3int), setlocale(3), scanf(3int), printf(3s), putc(3s), scanf(3s), stdio(3s)
Guide to Developing International Software

Name

scanf, fscanf, sscanf – convert formatted input

Syntax

#include <stdio.h>

int scanf(*format* [, *pointer*] ...)
char **format*;

int fscanf(*stream*, *format* [, *pointer*] ...)
FILE **stream*;
char **format*;

int sscanf(*s*, *format* [, *pointer*] ...)
char **s*, **format*;

Description

The international functions `scanf`, `fscanf`, and `sscanf` are similar to the `scanf` standard I/O functions. The difference is that the international functions allow you to use the *%digit$* conversion character in place of the *I%* character you use in the standard I/O functions. The *digit* is a decimal digit *n* from 1 to 9. The international functions apply conversions to the *n* th argument in the argument list, rather than to the next unused argument.

You can use % conversion character in the international functions. However, you cannot mix the % conversion character with the *%digit$* conversion character in a single call.

In all cases, `scanf` uses the radix character and collating sequence that is defined by the last successful call to `setlocale` category `LC_NUMERIC` or `LC_COLLATE`. If the radix or collating sequence is undefined, the `scanf` function uses the C locale definitions.

International Environment

LC_COLLATE Contains the user requirements for language, territory, and codeset for the character collation format. `LC_COLLATE` affects the behavior of regular expressions and the string collation functions in `scanf`. If `LC_COLLATE` is not defined in the current environment, `LANG` provides the necessary default.

LC_NUMERIC If this environment is set and valid, `scanf` uses the international language database named in the definition to determine radix character rules.

LANG If this environment variable is set and valid `scanf` uses the international language database named in the definition to determine collation and character classification rules. If `LC_NUMERIC` or `LC_COLLATE` is defined, their definitions supersede the definition of LANG.

Examples

The following shows an example of using the `scanf` function:

```
scanf("%2$s %1$d", integer, string)
```

If the input is " january 9 ", the `scanf` function assigns 9 to `integer` and "january" to `string`.

Return Values

The `scanf` function returns the number of successfully matched and assigned input fields. This number can be zero if the `scanf` function encounters invalid input characters, as specified by the conversion specification, before it can assign input characters.

If the input ends before the first conflict or conversion, `scanf` returns EOF. These functions return EOF on end of input and a short count for missing or invalid data items.

Environment

In POSIX mode, the **E**, **F**, and **X** formats are treated the same as the **e**, **f**, and **x** formats, respectively; otherwise, the upper-case formats expect double, double, and long arguments, respectively.

See Also

intro(3int), setlocale(3), strtod(3), strtol(3), printf(3int), getc(3s), printf(3s), scanf(3s)
Guide to Developing International Software

Name

vprintf, vfprintf, vsprintf – print formatted output of a varargs argument list

Syntax

```
#include <stdio.h>
#include <varargs.h>

int vprintf ( format, ap )
char *format;
va list ap;

int vfprintf ( stream, format, ap )
FILE *stream;
char *format;
va list ap;

int vsprintf ( s, format, ap )
char *s, *format;
va list ap;
```

Description

The international functions `vprintf`, `vfprintf`, and `vsprintf` are similar to the `vprintf` standard I/O functions.

Likewise, the vprintf functions are similar to the printf functions except they are called with an argument list as defined by `varargs` instead of with a variable number of arguments.

The international functions allow you to use the *%digit$* conversion character in place of the % character you use in the standard I/O functions. The digit is a decimal digit *n* from 1 to 9. The international functions apply conversions to the *nth* argument in the argument list, rather than to the next unused argument.

You can use the % conversion character in the international functions. However, you cannot mix the % conversion character with the *%digit$* conversion character in a single call.

You can indicate a field width or precision by an asterisk (*) instead of a digit string in format strings containing the % conversion character. If you use an asterisk, you can supply an integer *arg* that specifies the field width or precision. In format strings containing the *%digit$* conversion character, you can indicate field width or precision by the sequence **digit$*. You use a decimal digit from 1 to 9 to indicate which argument contains an integer that specifies the field width or precision.

The conversion characters and their meanings are identical to `printf`.

You must use each digit argument at least once.

Examples

```
#include <stdio.h>
#include <varargs.h>

main()
{
char *function_name = "vpr";
char *arg1 = "hello world";
int arg2 = 2;
char *arg3 = "study";

char *i18nfmt = "%1$s %3$d\n";

test(function_name, i18nfmt, arg1, arg2, arg3);
}

test(va_alist)
va_dcl
{
va_list args;
char *fmt;
char string[1024];

va_start(args);

(void)printf("function %s: ", va_arg(args, char *));

fmt = va_arg(args, char *);

(void)vprintf(fmt, args);

va_end(args);
}
```

See Also

setlocale(3), scanf(3int), printf(3s), printf(3int), vprintf(3s), putc(3s), scanf(3s), stdio(3s), varargs(3)
Guide to Developing International Software